Binary Search

//------lower bound ------

int find\_lowerBound(vector<int>&v , int value) {

int l = -1, r = v.size();

while (l + 1 < r) {

int m = (l + r ) / 2;

if (v[m] < value) {

l = m;

}else {

r = m;}

}return r;}

//------lower bound---------

int lower - bound(vector<int>&v, int value) {

int l = 1, r = n, m,ans=1e9+5;

while (l <= r) {

m = (l + r) / 2;

if (ok(v, m)) {

ans = min(ans, m);

r = m - 1;}

else { l = m + 1;}

}return ans;

//------upper bound---------

int find\_upperBound(vector<int>&v , int value) {

int l = -1, r = v.size();

while (l + 1 < r) {

int m = (l + r ) / 2;

if (v[m] > value) { r = m; }

else { l = m; } return r;

Bell man ford

//\*\*\*Bellman Ford will not work : if there exist a cycle with total weight is : negative

//after performing n time relaxation if n+1th relaxation value will change then it has cycle with neg value

int main()

{ struct Edge {

int u, v, weight;

};

Edge a, b, c, d, e, f;

/\*আমরা এজ লিস্ট তৈরি করছি, a,b,c,d,e,f যথা ক্রমে আমাদের বিভিন্ন এজ নির্দেশ করছে \*/

a.u = 0; a.v = 1; a.weight = 10;

b.u = 1; b.v = 2; b.weight = 1;

c.u = 2; c.v = 4; c.weight = 3;

d.u = 4; d.v = 3; d.weight = -11;

e.u = 3; e.v = 1; e.weight = 4;

f.u = 4; f.v = 5; f.weight = 22;

vector <Edge> E = {a, b, c, d, e, f};

/\*একটা অ্যারে নিতে হবে Cost যার সাইজ হবে মোট নোডের সংখ্যা। নামে, যেখানে আমরা প্রতিটি আপডেটের হিসাব রাখবো। প্রাথমিক ভাবে আমরা শুরুর নোড ০ বাদে সব নোডের মান INT\_MAX বা অসীম করে রাখবো \*/

int n = 6; int cost[n];

for (int i = 1; i < n; i++) {

cost[i] = INT\_MAX; }

cost[0] = 0; //শুরুর নোড এর মান শূন্য করে দিলাম

for (int i = 0; i < n - 1; i++) { /\*n-1 বার চলবে\*/

for (Edge edge : E) {

if (cost[edge.v] > cost[edge.u] + edge.weight) { //যদি নতুন কস্ট আগে রাখা কস্টের চেয়ে ছোট হয়, তবে

cost[edge.v] = cost[edge.u] + edge.weight; //আমাদের পুরাতন কস্টকে নতুন কস্ট দ্বারা আপডেট করতহবে।

}}}

//check negative cycle

for (Edge edge : E) {

if (cost[edge.v] > cost[edge.u] + edge.weight) { //যদি নতুন কস্ট আগে রাখা কস্টের চেয়ে ছোট হয়, তবে

cout << "Negetive cycle detected\n";break;}}

for (int i = 0; i < n; i++) { cout << "Distance of node " << i << " from node 0 is " << cost[i] << endl;}

BFS with path printing

void bfs(int src) {

queue<int>q; q.push(src) dist[src] = 0; parent[src] = src;

while (!q.empty()) {

int u = q.front();

q.pop();

for (auto v : gr[u]) {

if (dist[v] == INT\_MAX) {

dist[v] = dist[u] + 1;

parent[v] = u;

q.push(v);

}}}}

//store path direction

int src = 1, target = n, now = n;

path.pb(target);

while (now != src) {

now = parent[now];

path.pb(now)}

for (i = path.size() - 1; i >= 0; i--) cout << path[i] << sp;

Cycle in 2D

void dfs(int x, int y, int fromX, int fromY, char need){

if (x < 0 || x >= n || y < 0 || y >= m) return;

if (ch[x][y] != need) return; // diferent color

if (vis[x][y]) found = true; return;

vis[x][y] = 1;

int nextX, nextY;

for (int i = 0; i < 4; i++)

nextX = x + dx[i]; nextY = y + dy[i];

if (nextX == fromX && nextY == fromY)

continue;

dfs(nextX, nextY, x, y, need);

//call from main : dfs(i, j, 0, 0, ch[i][j]);

Cycle detect

void dfs(int u,int pr)

vis[u] = 1;

for(int v : gr[u])

{if(vis[v] == 0) dfs(v,u);

else if(vis[v] == 1 && v != pr) hasCycle = true;

Dijkstra

void dijkstra(int x, int y) {

// priority\_queue<pair<int, pair<int, int>>> pq;

priority\_queue<pair<int, pair<int, int>>, vector<pair<int, pair<int, int>>>, greater<pair<int, pair<int, int>>>> pq;

dist[x][y] = 0;

pq.push({0, {x, y}});

while (!pq.empty()) {

int p = pq.top().ss.ff;

int q = pq.top().ss.ss;

pq.pop();

for (int i = 0; i < 4; i++) {

int pp = p + dx[i];

int qq = q + dy[i];

if (pp >= 0 && pp < row && qq >= 0 && qq < col && cost[pp][qq] != 'X') {

int newDist = dist[p][q] + cost[pp][qq] - '0';

if (dist[pp][qq] > newDist) {

dist[pp][qq] = newDist;

pq.push({newDist, {pp, qq}});

}}}}}

Dijkstra single

void shortest\_path(int src, int dst)

{

vector<ll> dist(dst + 1, LLONG\_MAX);

vector<int> parent(dst + 1, -1);

dist[src] = 0;

// priority\_queue<pii, vector<pii>, greater<pii> >PQ;

priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;

pq.push({0, src}); //pq is pair first dist then source

while (!pq.empty())

{ auto pu = pq.top();

int u = pu.second;

pq.pop();

for (int i = 0; i < gr[u].size(); i++)

{

int v = gr[u][i];

int cst = cost[u][i];

if (dist[v] > dist[u] + cst)

{

parent[v] = u;

dist[v] = dist[u] + cst;

pq.push({dist[v], v});

}}}

vector<int> path;

int node = dst;

while (node != -1)

{

path.push\_back(node);

node = parent[node];

}

cin >> u >> v >> cst;

gr[v].push\_back(u);

gr[u].push\_back(v);

cost[v].push\_back(cst);

cost[u].push\_back(cst);

} shortest\_path(1, n);

DSU

int pr[MAX + 5];

int find\_parent(int u) {

if (u == pr[u]) return u;

return pr[u] = find\_parent(pr[u]);

}

void union\_set(int u, int v) {

int pu = find\_parent(u);

int pv = find\_parent(v);

pr[pu] = pv;

}

int query(int u, int v) {

int pu = find\_parent(u);

int pv = find\_parent(v);

return pu == pv; //if pu==pv then 1 else 0

}

for (i = 0; i < n; i++)

pr[i] = i;

union\_set(u, v); query(u, v)

Floyd warsal

void floydWarshall() {

for (int k = 1; k <= n; k++) {

for (int i = 1; i <= n; i++) {

for (int j = 1; j <= n; j++) {

cost[i][j] = min(cost[i][j], cost[i][k] + cost[k][j]);

/// main

cost[u][v] = min(cost[u][v], c);

cost[v][u] = min(cost[u][v], c);

floydWarshall();

BFS on Grid

void bfs(int x, int y) {

dist[x][y] = 0;

queue<pair<int, int>>q;

q.push({x, y});

parent[x][y] = -1;

while (!q.empty()) {

int a = q.front().first;

int b = q.front().second;

q.pop();

for (int i = 0; i < 4; i++) {

int aa = a + dx[i];

int bb = b + dy[i];

if (isValid(aa, bb)) {

dist[aa][bb] = dist[a][b] + 1;

parent[aa][bb] = i;

q.push({aa, bb});

//call from main : bfs(sx, sy);

DFS path printing

void dfs(int node, int pr) {

vis[node] = 1;

parent[node] = pr;

for (auto child : gr[node]) {

if (!vis[child]) dfs(child, node);

else if (vis[child] && child != pr) {

hasCycle = true;

ev = child;

sv = node;

return;

}

Topsort dfs

stack<int>stk;

void dfs(int u) {

vis[u] = 1;

for (auto child : graph[u]) {

if (!vis[child]) dfs(child);

} stk.push(u);

Topsort bfs

stack<int>stk;

std::vector<pair<int, int>> v;

priority\_queue<int>pt;

void bfs() {

for (int i = 1; i <= n; i++) {

if (inOrder[i] == 0) pt.push(-i);

}

while (!pt.empty()) {

int up = -pt.top();

pt.pop();

stk.push(up);

vec.pb(up);

for (auto child : graph[up]) {

inOrder[child]--;

if (inOrder[child] == 0) pt.push(-child);

vector sort by 2nd

//sort via pair sum

//if first 2 are same then via second

bool cmp(const pair<ll, ll>a, const pair<ll, ll>b)

if (a.first == b.first) return (a.second < b.second)

else return a.first > b.first;}

//sort by second element

bool comp(const pair<int, int> &a, const pair<int, int> &b ) {

return a.second > b.second;

Recursion

ll fact\_tailCall(ll n, ll result){

if(n==1) return result;

return fact\_tailCall(n - 1, (result \* n) % MOD);}

Segment tree

#define mid (start+end)/2

#define left start, mid, at+at

#define right mid+1, end, at+at+1

int tree[4 \* MAX + 5], arr[MAX + 5];

void build(int start, int end, int at) {

if (start == end) {

tree[at] = arr[start];

return;}

build(left);

build(right);

tree[at] = tree[at + at] + tree[at + at + 1];}

int query(int start, int end, int at, int l, int r) {

if (end < l || start > r) return 0;

if (start >= l && end <= r) return tree[at]; //full range valid

return query(left, l, r) + query(right, l, r); }

void update(int start, int end, int at, int index, int value) {

if (end < index || start > index) return;

if (start == end && start == index) {

tree[at] = value;

return;}

update(left, index, value);

update(right, index, value);

tree[at] = tree[at + at] + tree[at + at + 1];}

build(0, n - 1, 1); //(start,end,at) query(0, n - 1, 1, l, r); update(0, n - 1, 1, index, value);

void MergeSort(int \*arr,int L,int R){

if(L<R)

{ int M = L+(R-L)/2;

MergeSort(arr,L,M);

MergeSort(arr,M+1,R);

Sort(arr,L,M,R); }}

Sparse table

int table[20][MAX];

int ar[MAX], LOG[100005];

int n;

void sparse\_table() {

//LOG table generation

LOG[1] = 0;

for (int i = 2; i <= n; i++)

LOG[i] = 1 + LOG[i / 2];

//base : first row

for (int i = 0; i < n; i++) table[0][i] = ar[i];

for (int i = 1; i <= LOG[n]; i++) {

int length = 1 << i; //2^i

for (int j = 0; j + length <= n; j++) {

int a = table[i - 1][j];

int b = table[i - 1][j + (length / 2)]; //j theke leng pojjoto min

table[i][j] = min(a, b);}}}

//get min

int getMin(int l, int r) {

int power = LOG[r - l + 1]; //min koto power er pre calculate er ta use korbo

int pLen = 1 << power; //2^power

int a = table[p][l];

int b = table[p][r - pLen + 1];

return min(a, b);

//query(l,r) =min(table[k][l],table[k][r-l+1])}

sparse\_table();

getMin(l, r)

{"cmd": ["g++.exe","-std=c++14", "${file}", "-o", "${file\_base\_name}.exe", "&&" , "${file\_base\_name}.exe<input.txt>output.txt"],

"shell":true,

"working\_dir":"$file\_path",

"selector":"source.cpp"}